Internal Report 2012-2013-20 August 2013

Universiteit Leiden
Opleiding Informatica

Multi-scale map visualization

Erik Hollander

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Het is moeilijk om op digitale kaarten in een oogopslag alle details te zien die iemand nodig zou kunnen hebben. Een algoritme wordt voorgesteld om delen van een kaart in te kunnen zoomen zonder al te veel detail van de omgeving te verliezen. Dit gebeurt door de omgeving in te drukken om zo ruimte te maken voor het uitvergrote deel. Dit wordt gedaan via een treemap en een net van verbonden punten en het slim bepalen in welke richting nabijgelegen gebieden moeten uitvergroten.

Contents

1 Introductie

Als men hun reis op kaart wil zien, is het lastig om ieder detail te herkennen. De autoweg kan gevolgd worden, maar hoe iemand dan precies van de parkeerplaats naar de nabijgelegen eindbestemming komt, is niet altijd duidelijk. Hiervoor zijn kaarten met verschillende maten van details gemaakt. Echter is het onhandig om al deze kaarten los mee tenemen. Vandaar dat hier wordt onderzocht of het mogelijk is om een digitale kaart te maken waarbij gedeeltes van een map kunnen worden uitvergroot, om zowel de macro- als micro-details in een kaart te kunnen zien. Een voorbeeld van het probleem is in figuur 1 samen met een simpele representatie van een oplossing te zien. Het algoritme zelf zal de context uit de omgeving bewaren.

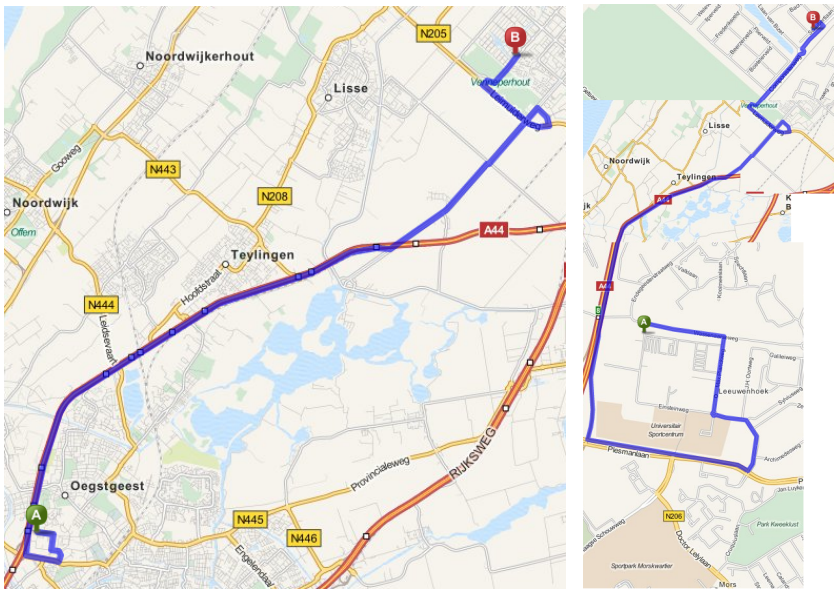

Figure 1: Aan de linkerzijde is het moeilijk te zien hoe de route in de stad loopt. Aan de rechterzijde is gemakkelijk te zien hoe de route loopt

In deze paper wordt een oplossing voorgesteld waarin gedeeltes van kaarten worden gemarkeerd om uitvergroot te worden, waarbij de omgeving van het gebied ingedrukt wordt. Dit betekent dat tegels in

deze omgeving zo vervormd worden dat ze zowel met het uitvergrote gebied als met de originele tegels verbinden. Dit zorgt ervoor dat bepaalde details uit een kaart naar boven worden gebracht zonder dat informatie uit de omgeving volledig verloren gaat. Dit wordt gedaan in de structuur van een simpele treemap waar alle punten uniek zijn. De tegel in een treemap bevat alle algemene informatie van belang voor het renderen van de kaart, zoals de texture en texture-coördinaten, terwijl de punten hun onaangepaste en huidige coördinaten bewaren. Deze methode heeft een vergelijkbaar resultaat met de rectangular fish-eye view[2][3], aangezien beide methoden de details naar voren halen, maar op andere manieren. De fish-eye method is alleen gemaakt om details naar voren te halen met als gevolg dat alles opzij geschoven moet worden om ruimte te maken. Dit is dan ook het belangrijkste verschil: bij de fish-eye method worden details kleiner naarmate ze verder van de focus liggen, terwijl bij deze methode hetgene wat het dichtste bij de focus ligt wordt vergroot, maar de omgeving hiervan minder detail krijgt toegewezen waarna de rest van de kaart onaangetast is.

De rest van deze paper is als volgt gestructureerd: in sectie 2 wordt een algoritme voorgesteld om dit probleem op te lossen. In sectie 3 wordt de implementie van het algoritme en javascript en webgl besproken en enkele resultaten getoond. In sectie 4 wordt de implementatie besproken en wat beter kan of waar later verder onderzoek naar kan worden gedaan.

2 Algoritme

In deze sectie wordt ingegaan op het algoritme, waar eerst wordt gekeken naar het algemene idee van het algoritme, dan de datastructuren die worden gebruikt, vervolgens de implementatie van de subdivide functie en uiteindelijk hoe het uitvergroten van elementen in de treemap gebeurt.

2.1 Algemene idee

Het algoritme gaat ervan uit dat een gebruiker een gebied aangeeft dat vergroot moet worden. Het algoritme bepaalt dan de omgeving die ingedrukt gaat worden om hier ruimte voor te maken. Deze gebieden worden eerst opgedeeld in de treemap in losse tegels. Vervolgens worden de tegels uitvergroot. Voor de omgeving wordt berekend hoe de tegels moeten vervormen om zowel op het vergrote gedeelte aan te sluiten als het onaangepaste uit de omgeving.

2.2 Datastructuren

Er zijn een viertal belangrijke datastructuren: de *pointNode*, de *tmTile*, de *magInfo* en de overkoepelende *treemap*.

PointNode

De *pointNode* bevat een punt, die een coördinaat op de kaart bevat, en een viertal variabelen die naar de pointNode direct ten noorden, oosten, zuiden en westen van deze node wijzen. Het punt bevat naast de huidige coördinaten op de kaart, ook de oorspronkelijke coördinaten en tijdelijke variabelen die tijdens het uitvergroten worden gebruikt om de nieuwe waarden in op te slaan voor ze permanent te maken. Ieder punt is uniek, dat wil zeggen, als de noord-west pointNode van een tegel punt X bevat, zal de noorderbuur, tenzij deze breder is, punt X in zijn zuid-west pointNode bevatten. Met als gevolg dat wanneer een punt wordt aangepast in een tegel, dit ook de buren zal aanpassen.

TmTile

De *tmTile* bevat een groot aantal variabelen die nodig zijn.

- parent
- Texture-coördinaten
- image
- nw
- ne
- se
- sw
- childTiles

De parent verwijst hier naar de ouder van de *tmTile*. De texturecoördinaten omschrijven het gedeelte van de texture die deze tegel beslaat. Image bevat een volledig mapsegment. De variabelen *nw*, *ne*,*se* en *sw* zijn pointNodes die de vier hoeken van de treemap omschrijven en childTiles is een matrix van *tmTiles*. Deze matrix is dynamisch aangewezen voor verschillende hoeveelheden tegels, maar wordt nooit groter dan 3 bij 3. Dit komt omdat in de meest drukke situatie een enkel gebied in een enkele tegel past, met ruimte aan alle kanten. In dit geval wordt het gebied opgedeeld in negen stukken. Er wordt nooit gekeken naar meerdere gebieden tegelijk, deze worden later opgedeeld in de nieuwe kinder-tegels.

MagInfo

Deze datastructuur beschrijft een gebied dat uitvergroot moet worden. Om precies te zijn bevat magInfo het volgende: een viertal punten die de uiteinden van het te uitvergroten gebied omschrijven, deze bevatten de beoogde locaties voor de uitvergroting en de waarden die het oorspronkelijk had op de kaart. Verder bevat het een tweetal punten die de grens van het gebied aangeeft dat ingedrukt moet worden. Deze datastructuur wordt in een lijst bewaard.

Treemap

Treemaps zijn in de jaren negentig geïntroduceerd voor het visueel weergeven van hiërarchische data[4] en zijn een logische te combineren met kaarten welke op verschillende niveaus verschillende data bevatten.

Een treemap is een boomstructuur waarin elke node typisch gezien een vierkant beschrijft en de kinderen van een node beschrijven dan een groep vierkanten die samen precies in het vierkant van de ouder passen. De grootte van een vierkant wordt bepaald door een bepaalde waarde, zoals het volume van wat omschreven wordt. Zo worden gerelateerde elementen bij elkaar gehouden en wordt het geheel visueel makkelijk te overzien. Een voorbeeld waar treemaps handig voor zijn, is de inhoud van een harde schijf visualiseren. De root stelt dan de volledige schijf voor en de kinderen de mappen op de harde schijf. De kinderen van de kinderen omschrijven dan weer submappen. Vierkanten worden groter naarmate de bestanden meer ruimte innemen. De treemap bevat een root *tmTile*, en functies voor het opdelen, vergroten en verkleinen van de tegels. Er valt hier verder weinig over te zeggen aangezien de meeste functionaliteit in de *tmTiles* zitten.

2.3 Subdivisie

De belangrijkste functie van de treemap zelf is het opdelen van de *tmTiles* in kleinere stukken. Echter is het handiger om eerst een kleinere functie die noodzakelijk is voor het opdelen te behandelen. Deze functie, algorithm 1, wordt voor het bepalen van de orientatie bij het noorden getoond. De andere orientaties volgen logisch uit dit voorbeeld.

Algorithm 1 Beslis orientatie Noord

```
Functie: beslisOrientatieNoord
Input: huidige tmTile T, vierkant X die in de treemap moet worden gedeeld
Output: twee variabelen noord en eendimNoord, allebei true of false
Algoritme:
noord = false
eenDimNoord = false
if noordlijn X zich tussen T's nw en zw punten bevindt then
    if noordlijn X ten westen van T begint then
        if noordlijn X voorbij westlijn T gaat then
            noord = true
        end if
    end if
else
    if noordlijn X niet oostlijn T voorbij gaat then
        noord = true
    end if
    eenDimNoord = !noord
end if
```

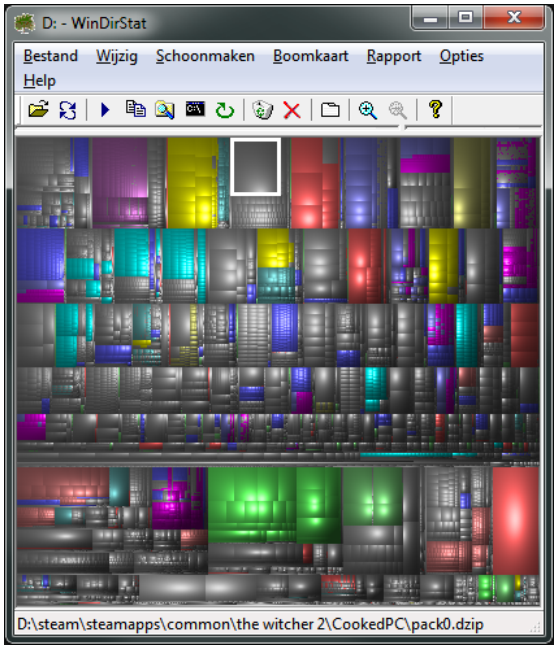

Figure 2: Een voorbeeld treemap die de inhoud van een harde schijf omschrijft.[5]

Nu het algoritme weet welke zijden van een gebied aangegeven door magInfo door de huidige tegel lopen, is het logisch te bepalen op welke manier de childtiles moeten worden ingedeeld, wat wordt getoond in algorithm 2. Er is een andere versie van het subdivide algoritme speciaal voor wanneer het vierkant wordt ingedeeld dat uitvergroot moet worden, dit heeft in de laatste fase met de eenDim variabelen een ander effect, want dan wordt niet gekeken of de *tmTile* al eerder is opgedeeld. Figuur 3 is een voorbeeld van een mogelijk subdivisie die gedaan moet worden. Eerst wordt gekeken bij tegel 1. Algorithm 1 geeft aan dat de noordlijn door de tegel heengaat. De andere algoritmen hiervan afgeleid tonen vervolgens aan dat ook de west en zuid zijde door de tegel heenlopen. Met deze combinatie van zijden die de tegel snijden, is het duidelijk de bedoeling om de tegel in zessen op te delen, met 2 colommen en 3 rijen. Hetzelfde geld voor tegel 2, al zijn de verhouding dan wat anders. Aan de rechterkant is de uiteindelijke tegelverdeling te zien.

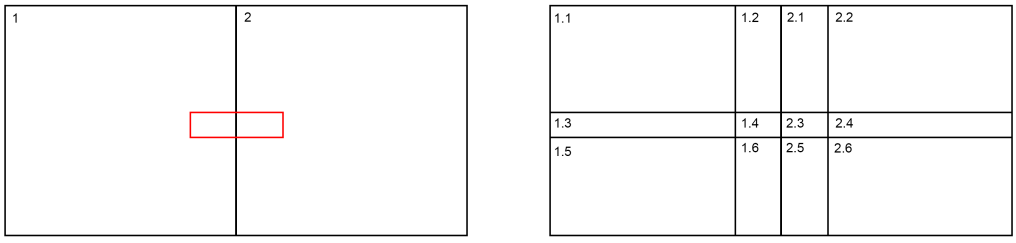

Figure 3: Twee tegels die het rode vierkant moeten indelen. Rechts is de oplossing te zien.

Bij het opdelen van de treemap krijgen alle kinderen hun nieuwe texture-coördinaten en punten. Ieder punt dat wordt aangemaakt, wordt eerst gekeken of deze niet al bestaat als onderdeel van een andere tegel. Als dit het geval is, wordt dit punt gebruikt in plaats van een nieuwe aan te maken.

2.4 Uitvergroten

Het uitvergroten van elementen in de treemap gaat in vier fasen. In de eerste fase wordt de gebieden die uitvergroot worden bepaald. In de tweede fase worden de tegels zelf vergroot, en wordt beslist hoe de direct aangelegen tegels ingedrukt worden. In de derde fase worden de tegels in die niet direct eraan zijn gelegen ingedrukt, en in de laatste fase worden de uiteindelijke nieuwe waarden van alle

Algorithm 2 subdivide

```
if childTiles is niet null then
    subdivide childTiles
end if
beslisOrientatie
if noord, zuid, oost, west niet allen false zijn then
    if noord en oost en zuid en west true zijn then
        deel tmTile op in negen stukken
    end if
    if 3 van noord, oost, zuid, west true zijn then
        deel tmTile op in 6 stukken
    end if
    if 2 van noord, oost, zuid, west true zijn then
        if (noord en zuid) of (oost en west) then
            deel tmTile op in 3 stukken
        else
            deel tmTile op in 4 stukken
        end if
        if 1 van noord, oost, zuid, west true is then
            deel tmTile op in 2 stukken
        end if
    end if
else
    if eenDimNoord, eenDimOost, eenDimZuid, eenDimWest niet allen false then
        if tmTile al eerder is opgedeeld voor een vergroting then
            if (eenDimNoord en eenDimZuid) of (eenDimOost en eenDimWest) then
                deel tmTile op in 3 stukken
            end if
            if 1 van eenDimNoord, eenDimOost, eenDimZuid, eenDimWest is true then
                deel tmTile op in 2 stukken
            end if
        end if
    else
        if tmTile wordt omvat door vierkant then
            zet variabelen zodat tmTile herkend wordt als onderdeel van de vergroting
        end if
    end if
end if
```

punten vastgezet.

Fase 1, voorbereidend werk

Als eerste wordt bepaald wat uitvergroot wordt, en hoe. De gebruiker geeft gebieden aan die uitvergroot moeten worden. Mocht de standaard manier van uitvergroten in alle richtingen zorgen voor overlap tussen twee gebieden, wordt besloten hoe de twee een bepaalde richting op kunnen uitvergroten om elkaar niet in de weg te zitten. Dit wordt gedaan met behulp van de Separating Axis Theorem[1] die voor de x- en y-axis een overlap vector genereert en als beide axes een vector hebben, overlapt deze. De gebieden worden dan uit elkaar gehaald door te kijken naar het kleinste component van de vector, en in die richting worden ze gescheiden. Hierna worden eventuele vergrotingen al op de map ongedaan gemaakt. Dan wordt de map eerst opgedeeld om ruimte te maken voor de gebieden die ingedrukt moeten worden en daarna voor de gebieden uitvergroot moeten worden.

Fase 2, uitvergroten

De tweede fase gaat voor iedere vergroting één keer alle tegels door. Als een tegel onderdeel blijkt te zijn van een gebied dat vergroot moet worden, wordt deze uitvergroot aan de hand van de eerder bepaalde nieuwe locatie in fase 1. Deze fase wordt tegelijk uitgevoerd met fase 3, maar is belangrijk genoeg om apart te houden. De nieuwe locaties van een tegel worden bepaald door de informatie in de magInfo die bij de tegel hoort. Voor ieder punt wordt een percentage berekend die aangeeft hoe ver een punt zich op de x- of y-as bevindt tussen twee grensaangevende punten van de magInfo. Deze verhouding moet bewaard blijven, dus het percentage wordt dan gebruikt om de nieuwe waarde tussen de uitvergrote punten van de magInfo te berekenen. Het is verder aan te merken dat voor ieder punt uiteindelijk een gemiddelde waarde wordt ingesteld in de laatste fase, dus als een andere vergroting door het gebied heenloopt, heeft dit licht effect op de anders perfect rechthoekige vorm van de vergroting. Om dit te beperken telt een vergroting van een punt veel zwaarder in het berekenen van het gemiddelde dan andere punten.

Fase 3, eerste tegels indrukken

De derde fase gaat de eerste batch tegels door om in te drukken. Als een gebied ingedrukt moet worden, wordt gekeken naar de oriëntatie van het gebied ten opzichte van het vergrote gebied. Dit omdat gebieden in diagonale richtingen niet triviaal te berekenen zijn. Alleen gebieden direct ten noorden, oosten, zuiden en westen worden aangepast door te berekenen op hoeveel procent het punt zit tussen de oorspronkelijke waarde van de grens van het uitvergrootte gebied, en de grens van het gebied dat ingedrukt wordt. Hiermee wordt de correcte locatie tussen de grens van het uitvergrootte gebied en de grens van het indruk-gebied berekend. In figuur 4 wordt getoond hoe de nieuwe coördinaten worden berekend.

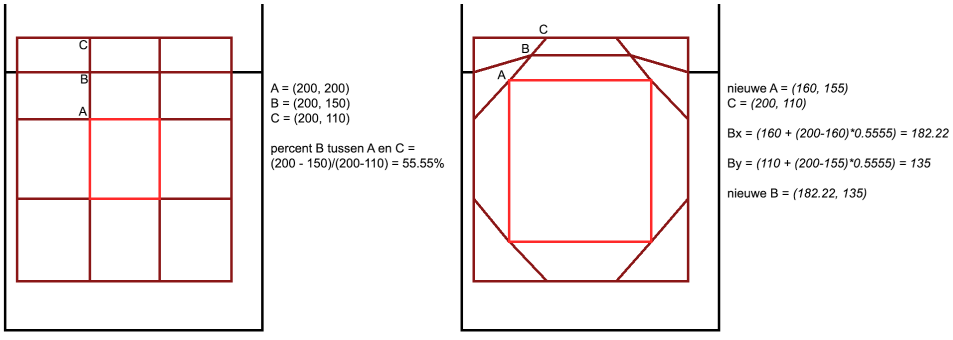

Figure 4: Het indrukken van tegels. Hier wordt getoond hoe B berekend wordt aan de hand van de gegevens uit punt A en C, en het hoe B tussen deze twee in ligt.

Fase 4, de rest van de nabijgelegen tegels indrukken

De vierde fase drukt de overige tegels, die ingedrukt moeten worden, in. Wat er gebeurt is dat, net zoals in fase 3 de tegels direct ten noorden, oosten, zuiden, of westen te vinden, en van daaruit via de pointNode verbindingen de tegels te behandelen die niet direct in een windrichting liggen. Zo wordt

bij een tegel ten noorden van het uitvergrootte gedeelte, als de noordwest punt van de tegel aan de rand van de windrichting ligt, een punt naar het westen opgeschoven. Hier wordt berekend hoe ver dit punt zich bevindt tussen het punt waar het algoritme vandaan komt en de rand van het gebied dat ingedrukt moet worden. Deze waarde wordt dan gebruikt om een y-waarde te berekenen die tussen de locatie van het noordelijke punt van de uitvergrote tegel en de oorspronkelijke locatie in ligt. Deze wordt vervolgens opgeslagen als nieuwe y-waarde. Als er al eerder een y-waarde is opgeslagen voor dit punt, wordt het gemiddelde tussen de twee genomen. Dit gebeurt ook voor alle andere punten.

Fase 5, definitieve waarden vastzetten

De laatste fase neemt de berekende waarden uit de vorige twee fasen en past de waarden toe op de punten. Pas vanaf dit moment zal het te zien zijn in de gerenderde map. Er is besloten om het niet eerder vast te zetten omdat dit extra overhead oplevert. Het zou kunnen na de eerste fase, maar alle tegels worden nagelopen, wat een redelijke complexiteit heeft. Het is dan beter om alles in een keer in te stellen. Mocht tijdens het instellen worden ontdekt bij een punt dat bijvoorbeeld zijn noorderbuur ten zuiden ligt, betekent dit ongewenste overlap. In dat geval worden alle vergrotingen ongedaan gemaakt, en wordt teruggegaan naar fase een zodat de gebieden uit elkaar gedrukt kunnen worden en het opnieuw kan gebeuren.

alles samen

Het uiteindelijke algoritme die al deze fasen uitvoert staat vervolgens in algoritme 3 en 4. Het eerste algoritme is de voorbereidende fase, terwijl het tweede algoritme de overige fasen gebruikt. De tweede en derde fase gebeuren samen in de functie this.root.magnify, terwijl fase vier gebeurt in de functie magnifyDiagonals. De laatste fase wordt gedaan door de functies setNewCoords, en in algoritme 3 wordt hier verder op ingegaan wanneer de functie false blijkt te returnen. In dat geval wordt de separateMags functie aangeroepen en wordt het algoritme opnieuw aangeroepen om alles overnieuw te doen. De functie separateMags zorgt ervoor dat gebieden die te dicht in de buurt van elkaar liggen, uit elkaar worden geduwd zodat ze niet meer over elkaar vallen. De functie cleanup in algoritme 4 zorgt ervoor dat variabelen weer aangepast kunnen worden. Het is niet wenselijk wanneer de nieuwe coördinaten berekend worden aan de hand van n vergrotingsgebied, dat deze meerdere keren wordt aangepast, dit zou een ongewenst effect hebben op het berekenen van de gemiddelden. Zodra de coordinaten worden berekend wordt het gemiddelde opgeslagen en een label aan het punt toegewezen dat aangeeft dat deze niet meer aangepast mag worden. De functie cleanUp verwijdert deze labels weer, waarna het gemiddelde van het punt weer kan worden aangepast.

Algorithm 3 mapStretch

```
Functie: mapStretch
Input: lijst informatie gebieden die uitvergroot moeten worden, magInfoList, treemap tm
Algoritme:
undoMagnify()
detectCollission()
for i = 0; i ≤ magInfoList.length; i++ do
    for j = ij ≥ 0; j++ do
        subdivide (magInfoList[j].squashArea)
        subdivide (magInfoList[j].magArea)
    end for
end for
if !tm.magnify(magInfoList) then
    separateMags
    mapStretch(magInfoList, tm)
end if
```

3 Implementatie

Voor het algoritme is een simpele website aangemaakt/refmapstretch die dit algoritme implementeert op een vak van 3 bij 3 kaart-tegels. Dit is gedaan met webgl voor het renderen van de kaart en javascript voor de implementatie van het algoritme. Daarnaast moest een simpele shader in GLSL geschreven worden om de kaart in 2D te tekenen. Al snel werd bekend dat in deze implementatie een

Algorithm 4 tm.magnify

```
Functie: tm.magnify
Input: lijst informatie gebieden die uitvergroot moeten worden
Algoritme:
for i = 0; i ≤ magInfoList.length; i++ do
    this.root.magnify(magInfoList, i)
    this.root.cleanUp
end for
for i = 0; i ≤ magInfoList.length; i++ do
    this.root.magnifyDiagonals(magInfoList, i)
    this.root.cleanUp
end for
if !this.root.setNewCoords then
    this.undoMagnify()
    return false
end if
return true
```

node nooit meer dan drie kinderen zou krijgen, dus is besloten om deze kinderen in een matrix op te slaan. De breedte en hoogte van deze matrix wordt dynamisch aangemaakt om zo de verdeling van tegels te vertegenwoordigen. Om te zorgen dat alle punten uniek en verbonden zijn, moet wanneer nieuwe punten worden aangemaakt tijdens het opdelen van de tegels, wordt gekeken of deze niet al eerder zijn aangemaakt. Als het punt al eerder in het net van punten is geplaatst, dan moet dit punt worden toegewezen in plaats van het nieuw aangemaakte punt. Hierna worden de punten met elkaar verbonden. Tijdens het opdelen van een tegel wordt een matrix van alle punten, die hierbij betrokken zijn, gemaakt. In deze matrix is het dan erg simpel om te bepalen wat de buren zijn.

Voor het berekenen van nieuwe coördinaten voor punt X wordt, zoals in fase 3 en 4 wordt omschreven, berekend hoever dit punt zich tussen twee coördinaten bevindt. Van deze coördinaten ligt de ene, *magCoord*, op de grens van het te vergroten gebied en de ander, *origCoord*, op de grens van het in te drukken gebied. X ligt natuurlijk tussen deze twee coördinaten. De coordinaten van X worden dan berekend met het percentage dat in het begin is gevonden. X moet tussen de uitvergrote coördinaten in *magCoord* en de coördinaten in *origCoord* liggen, met dezelfde verhoudingen als in het begin.

Voor het opslaan van de berekende nieuwe coördinaten wordt bij ieder punt een aantal extra variabelen toegewezen. Dankzij javascript is het gemakkelijk om tijdelijke variabelen aan een object toe te voegen. De tijdelijke variabelen bewaren data over de gemiddeld berekende waarde. Er is een variabele die het huidige gemiddelde bewaard en een variabele die bijhoudt hoeveel waarden zijn toegevoegd aan het gemiddelde. Door het huidige gemiddelde te vermenigvuldigen met hoeveel waarden aan het gemiddelde zijn toegevoegd, krijgt het algoritme dezelfde waarde als wanneer alle waarden zouden worden opgesomd. Op deze manier hoeft minder data bewaard te worden. Wanneer het algoritme klaar is, worden de tijdelijke variabelen weer uit het object verwijderd om ruimte te besparen.

Resultaten

In Figuur 5 wordt een voorbeeld getoond van het programma waarin een enkel gebied wordt uitvergroot. Hier valt te zien dat het gebied dat ingedrukt gaat worden, dezelfde verhoudingen heeft als het gebied dat uitvergroot wordt. Om precies te zijn de breedte en hoogte van het roze gebied drie maal zo groot als die van het rode gebied. Het resultaat hiervan is te zien aan de rechterkant.

Figuur 6 toont een voorbeeld dat twee aangelegen stukken uitvergroot. Het valt hier op hoe het roze gebied niet langer dezelfde proporties heeft als het rode gebied. Dit komt omdat de twee gebieden moeten uitwijken om niet over elkaar heen getekend te worden. Aan de rechterzijde is te zien hoe dat eruit ziet.

4 Conclusie

Het algoritme werkt en vergroot de omgevingen zonder context te verliezen. Het gebied dat ingedrukt wordt is niet altijd even duidelijk en visuele optimalisatie is mogelijk. Dingen om over na te denken zijn wat dit betreft straatnamen die niet op de kaart worden meevergroot, maar dynamisch bovenop

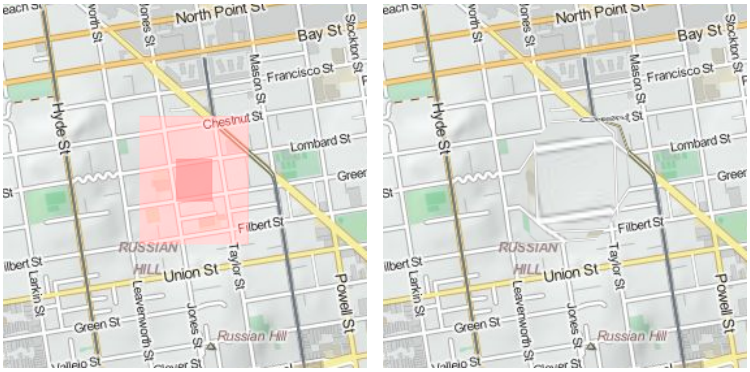

Figure 5: Een enkel gebied voor en na vergroting

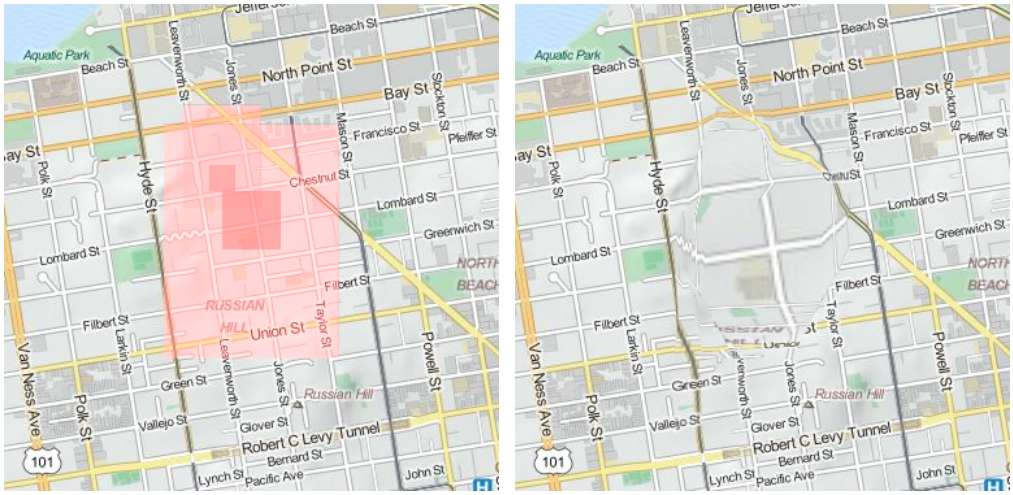

Figure 6: Een tweetal gebieden voor en na vergroting

de kaart worden gerenderd. Ook zijn sommige vervorming van tegels moeilijker om goed te renderen en een algoritme die de textures mooier kan spreiden over de vervormde polygonen zou gewenst zijn. Er zijn andere manieren om delen van kaarten uit te vergroten, zoals het gescheiden houden van alle tegels, om op die manier ruimte te maken voor vergroting. Dit zou als gevolg hebben dat tegels niet ingedrukt hoeven te worden om ruimte te maken. Echter neemt deze manier uiteindelijk veel ruimte in op het scherm, en wordt overzicht verloren, zoals hoe de wegen op elkaar aansluiten. Ook zou het mogelijk zijn om overlap te accepteren en als het ware vergrootglazen op de map te kunnen plaatsen. Dit zijn in de juiste omstandigheden goede mogelijkheden, maar het verliezen van context is niet altijd gewenst en daar is dit algoritme een antwoord op.

Discussie

De bedoeling van het algoritme is om stukken van de map te vergroten zonder de context uit de omgeving te verliezen. De rekentijd van het programma neemt wel snel toe naarmate de hoeveelheid gebieden die uitvergroot moeten worden, toeneemt. Op de test-PC duurde het vergroten van 10 gebieden meerdere seconden, en op veel PCs zal dat nog langzamer gaan. Hierdoor is het voor gemiddeld gebruik, bijvoorbeeld om een hele route uit te vergroten, niet gebruikersvriendelijk. Het doorzoeken van de treemap voor tegels wordt recursief gedaan en genereert snel overhead. Veel tegels zullen altijd onnodig zijn om ook maar iets mee te doen. De treemap structuur is niet optimaal. Het doet zijn dienst om belangrijke informatie te bewaren, maar sommige functies hebben een hoge complexiteit, met name functies waarin in de treemap moet worden gezocht. Het zou interessant zijn om uit te zoeken of een platte datastructuur gebruikt zou kunnen worden om benodigde informatie bij te houden. Anderzijds kan ook gekeken worden naar manieren om het doorzoeken van de treemap te optimaliseren, bijvoorbeeld door gerichter op zoek te gaan naar tegels die nodig zijn. De kinderen van de root zijn altijd identiek dus het zou mogelijk zijn om een functie te maken die bekijkt in

welke tegel gezocht moet worden naar een bepaald element. Dit zou al aardig wat schelen met name wanneer alle tegels veel zijn opgedeeld. Alhoewel het algoritme goed genoeg moet kunnen omgaan met het verkleinen van stukken in de map, is hier niet op getest en dat zou een interessant onderwerp zijn om verder op in te gaan. In deze specifieke implementatie was het renderen van de map met weinig aandacht gedaan waardoor sommige tegels slecht getekend worden bij bepaalde vergrotingen. Als een tegel zo wordt ingedrukt dat deze op een driehoek lijkt, waarbij de schuine zijde van het zuidwesten naar het noordoosten gaat, wordt de ene helft van de texture nauwelijks tot niet getoond, terwijl de andere helft van de texture lijkt alsof het nergens door is aangetast. Een algoritme dat dit soort polygonen beter kan renderen zou gewenst zijn.

5 References

[1] S. Gottschalk. *Separating axis theorem.* 1996.

[2] Doug Schaffer, Zhengping Zuo, Lyn Bartram, John Dill, Shelli Dubs, Saul Greenberg, Mark Roseman *Comparing Fisheye and Full-Zoom Techniques for Navigation of Hierarchically Clustered Networks.* 1993

[3] Priya Sundararajan, Ole J. Mengshoel, Ted Selker *Multi-Fisheye for Interactive Visualization of Large Graphs.* 2011

[4] B. Johnson, B. Shneiderman. *Tree-Maps: a space-filling approach to the visualization of hierarchical information structures* 1991

[5] *http://windirstat.info/*

[6] *http://www.gerbebras.nl/mapstretch*

Appendices

Vergroting

De hieronder volgende code toont de belangrijkste functies voor het vergroten en indrukken van gebieden in de code. De functie *tmTile.prototype.magnify* is de implementatie van fase 2 en 3 zoals beschreven in sectie 2.4. De functie *tmTile.prototype.magnifyDiagonals* is de implementatie van fase 4. De functies *calcNewPointCoordsEast, calcNewPointCoordsSouth en calcNewPointCoordsWest* zijn niet getoond, maar volgen logisch uit de functie *calcNewPointCoordsNorth*. De *magInfoList* is een globale lijst die instanties van *magInfo* bevatten. Deze datastructuur is uitgelegd in sectie 2.2. De functie *CleanUp* zorgt ervoor dat ieder punt weer opnieuw aangepast kan worden. Dit om te voorkomen dat punten meermalig aangepast worden voor een enkel gebied, wat het gemiddelde berekenen zou beïnvloeden later.

```
1  function magnify() {
2    tm.undoMagnify();
3    for (var i = 0; i < current; i++) {
4      for (var j = i; j >= 0; j--) {
5        tm.subdivide(magInfoList[j].squashnw, magInfoList[j].squashse,
             magInfoList[j].magnw, magInfoList[j].magse);
6        tm.magSubdivide(magInfoList[j].magnw, magInfoList[j].magse, 2);
7        render();
8        magInfoList[j].magnified = true;
9      }
10   }
11   if (!tm.magnify(magInfoList)) {
12     separateMags();
13     magnify();
14   }
15 }
16
17 treemap.prototype.magnify = function(magInfoList) {
18   for (var i = 0; i < magInfoList.length; i++) {
19     this.root.magnify(magInfoList, i);
20     this.root.cleanUp();
21   }
22   for (var i = 0; i < magInfoList.length; i++) {
23     this.root.magnifyDiagonals(magInfoList, i);
24     this.root.cleanUp();
25   }
26   if (!this.root.setNewCoords()) {
27     this.undoMagnify();
28     return false;
29   }
30   return true;
31 }
32
33
34 tmTile.prototype.magnify = function (magInfoList, magInfoNum) {
35   if (this.childTiles != null) {
36     for (var i = 0; i < this.childTiles.length; i++) {
37       for (var j = 0; j < this.childTiles[i].length; j++) {
38         this.childTiles[i][j].magnify(magInfoList, magInfoNum);
39       }
40     }
41   } else {
42
43     if (this.scale !== undefined) {
44       if (magInfoList[magInfoNum].magContains(this)) {
45         var percent = (this.nw.point.ox - magInfoList[magInfoNum].magnw.ox)/
               magInfoList[magInfoNum].width;
46         setNewCoords(this.nw.point, calcNewCoord(magInfoList[magInfoNum].magnw.
               x, magInfoList[magInfoNum].magne.x, percent), null, 8);
47         setNewCoords(this.sw.point, calcNewCoord(magInfoList[magInfoNum].magsw.
               x, magInfoList[magInfoNum].magse.x, percent), null, 8);
48
49         percent = (this.ne.point.ox - magInfoList[magInfoNum].magnw.ox)/
               magInfoList[magInfoNum].width;
50         setNewCoords(this.ne.point, calcNewCoord(magInfoList[magInfoNum].magnw.
               x, magInfoList[magInfoNum].magne.x, percent), null, 8);
```

```
51          setNewCoords(this.se.point, calcNewCoord(magInfoList[magInfoNum].magsw.
                x, magInfoList[magInfoNum].magse.x, percent), null, 8);
52
53          percent = (this.ne.point.oy - magInfoList[magInfoNum].magne.oy)/
                magInfoList[magInfoNum].height;
54          setNewCoords(this.nw.point, null, calcNewCoord(magInfoList[magInfoNum].
                magnw.y, magInfoList[magInfoNum].magsw.y, percent), 8);
55          setNewCoords(this.ne.point, null, calcNewCoord(magInfoList[magInfoNum].
                magne.y, magInfoList[magInfoNum].magse.y, percent), 8);
56
57          percent = (this.se.point.oy - magInfoList[magInfoNum].magne.oy)/
                magInfoList[magInfoNum].height;
58          setNewCoords(this.sw.point, null, calcNewCoord(magInfoList[magInfoNum].
                magnw.y, magInfoList[magInfoNum].magsw.y, percent), 8);
59          setNewCoords(this.se.point, null, calcNewCoord(magInfoList[magInfoNum].
                magne.y, magInfoList[magInfoNum].magse.y, percent), 8);
60        }
61      } else if (this.squash == true) {
62        var index = null;
63        if (magInfoList[magInfoNum].contains(this)) {
64          var index = magInfoNum;
65          var north = false, east = false, south = false, west = false;
66
67          if (this.nw.point.oy < magInfoList[magInfoNum].magnw.oy) {
68            north = true;
69          } else if (this.sw.point.oy > magInfoList[magInfoNum].magse.oy) {
70            south = true;
71          }
72          if (this.nw.point.ox < magInfoList[magInfoNum].magnw.ox) {
73            west = true;
74          } else if (this.ne.point.ox > magInfoList[magInfoNum].magse.ox) {
75            east = true;
76          }
77          if (north && west) {
78          } else if (north && east) {
79          } else if (south && east) {
80          } else if (south && west) {
81          } else if (north) {
82            calcNewPointCoordsNorth(this.nw.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magnw);
83            calcNewPointCoordsNorth(this.ne.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magne);
84            calcNewPointCoordsNorth(this.se.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magne);
85            calcNewPointCoordsNorth(this.sw.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magnw);
86          } else if (east) {
87            calcNewPointCoordsEast(this.nw.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magne);
88            calcNewPointCoordsEast(this.ne.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magne);
89            calcNewPointCoordsEast(this.se.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magse);
90            calcNewPointCoordsEast(this.sw.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magse);
91
92          } else if (south) {
93            calcNewPointCoordsSouth(this.nw.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magsw);
94            calcNewPointCoordsSouth(this.ne.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magse);
95            calcNewPointCoordsSouth(this.se.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magse);
96            calcNewPointCoordsSouth(this.sw.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magsw);
97
98          } else if (west) {
99            calcNewPointCoordsWest(this.nw.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magnw);
100           calcNewPointCoordsWest(this.ne.point, magInfoList[magInfoNum],
                  magInfoList[magInfoNum].magnw);
```

```
101             calcNewPointCoordsWest(this.se.point, magInfoList[magInfoNum],
                    magInfoList[magInfoNum].magsw);
102             calcNewPointCoordsWest(this.sw.point, magInfoList[magInfoNum],
                    magInfoList[magInfoNum].magsw);
103           }
104         }
105       }
106     }
107 }
108
109 function calcNewPointCoordsNorth(point, magInfo, currentLoc) {
110   if (shouldBeChanged(point, magInfo.magnw, magInfo.magse, magInfo.squashnw,
        magInfo.squashse)) {
111     var percent = (magInfo.magnw.oy - point.oy)/(magInfo.magnw.oy - magInfo.
          squashnw.oy);
112     var xPercent = (point.ox - magInfo.magnw.ox)/magInfo.width;
113     var currentX = calcNewCoord(magInfo.magnw.x, magInfo.magne.x, xPercent);
114     var destX = calcNewCoord(magInfo.magnw.ox, magInfo.magne.ox, xPercent);
115     setNewCoords(point, calcNewCoord(currentX, destX, percent), calcNewCoord(
          currentLoc.y, magInfo.squashnw.y, percent));
116   } else {
117     if (point.oy != currentLoc.oy) {
118       setNewCoords(point, point.ox, point.oy);
119     }
120   }
121 }
122
123 function calcNewCoord(e, o, percent) {
124   return e + ((o - e)*percent);
125
126 }
127 tmTile.prototype.magnifyDiagonals = function (magInfoList, magInfoNum) {
128   if (this.childTiles != null) {
129     for (var i = 0; i < this.childTiles.length; i++) {
130       for (var j = 0; j < this.childTiles[i].length; j++) {
131         this.childTiles[i][j].magnifyDiagonals(magInfoList, magInfoNum);
132       }
133     }
134   } else {
135     if (this.squash) {
136       var index = null;
137       var north = false, east = false, south = false, west = false;
138
139       if (magInfoList[magInfoNum].contains(this)) {
140         var pn;
141         var percent;
142         north = false;
143         east = false;
144         south = false;
145         west = false;
146         if (this.nw.point.oy < magInfoList[magInfoNum].magnw.oy) {
147           north = true;
148         } else if (this.sw.point.oy > magInfoList[magInfoNum].magse.oy) {
149           south = true;
150         }
151         if (this.nw.point.ox < magInfoList[magInfoNum].magnw.ox) {
152           west = true;
153         } else if (this.ne.point.ox > magInfoList[magInfoNum].magse.ox) {
154           east = true;
155         }
156         if (north && west) {
157         } else if (north && east) {
158         } else if (south && east) {
159         } else if (south && west) {
160         } else if (north) {
161           if (this.nw.point.ox == magInfoList[magInfoNum].magnw.ox) {
162             pn = this.nw.west;
163             while (pn !== undefined && pn.point.ox >= magInfoList[magInfoNum].
                    squashnw.x) {
164               percent = (magInfoList[magInfoNum].magnw.ox - pn.point.ox)/
                      magInfoList[magInfoNum].width;
```

```
165                 setNewCoordsDiag(pn.point, null, (this.nw.point.yChange + (this.
                        nw.point.oy - this.nw.point.yChange)*percent));
166                 pn = pn.west;
167               }
168             }
169
170             if (this.ne.point.ox == magInfoList[magInfoNum].magse.ox) {
171               pn = this.ne.east;
172               while (pn !== undefined && pn.point.ox <= (magInfoList[magInfoNum].
                      squashse.x)){
173                 percent = (abs(magInfoList[magInfoNum].magse.ox - pn.point.ox))/
                        magInfoList[magInfoNum].width;
174                 setNewCoordsDiag(pn.point, null, (this.ne.point.yChange - (this.
                        ne.point.yChange - this.ne.point.oy)*percent));
175                 pn = pn.east;
176               }
177             }
178
179           } else if (east) {
180             if (this.nw.point.oy == magInfoList[magInfoNum].magnw.oy) {
181               pn = this.ne.north;
182               while (pn !== undefined && pn.point.oy >= (magInfoList[magInfoNum].
                      squashnw.y)) {
183                 percent = (abs(magInfoList[magInfoNum].magnw.oy - pn.point.oy))/
                        magInfoList[magInfoNum].height;
184                 setNewCoordsDiag(pn.point, (this.ne.point.xChange + (this.ne.
                        point.ox - this.ne.point.xChange)*percent), null);
185                 pn = pn.north;
186               }
187             }
188
189             if (this.sw.point.oy == magInfoList[magInfoNum].magse.oy) {
190               pn = this.se.south;
191               while (pn !== undefined && pn.point.oy <= (magInfoList[magInfoNum].
                      squashse.y)) {
192                 percent = (abs(magInfoList[magInfoNum].magse.oy - pn.point.oy))/
                        magInfoList[magInfoNum].height;
193                 setNewCoordsDiag(pn.point, (this.se.point.xChange - (this.se.
                        point.xChange - this.se.point.ox)*percent), null);
194                 pn = pn.south;
195               }
196             }
197
198           } else if (south) {
199             var destY = this.sw.point.oy;
200             if (this.se.point.ox == magInfoList[magInfoNum].magse.ox) {
201               pn = this.se.east;
202               while (pn !== undefined && pn.point.ox <= (magInfoList[magInfoNum].
                      squashse.x)){
203                 percent = (abs(magInfoList[magInfoNum].magse.ox - pn.point.ox))/
                        magInfoList[magInfoNum].width;
204                 setNewCoordsDiag(pn.point, null, calcNewCoord(this.se.point.
                        yChange, destY, percent));
205                 pn = pn.east;
206               }
207             }
208
209             if (this.sw.point.ox == magInfoList[magInfoNum].magnw.ox) {
210               pn = this.sw.west;
211               while (pn !== undefined && pn.point.ox >= (magInfoList[magInfoNum].
                      squashnw.x)){
212                 percent = (magInfoList[magInfoNum].magnw.ox - pn.point.ox)/
                        magInfoList[magInfoNum].width;
213                 setNewCoordsDiag(pn.point, null, calcNewCoord(this.sw.point.
                        yChange, destY, percent));
214                 pn = pn.west;
215               }
216             }
217
218           } else if (west) {
219             var destX = this.nw.point.ox;
```

```
220             if (this.nw.point.oy == magInfoList[magInfoNum].magnw.oy) {
221               pn = this.nw.north;
222               while (pn !== undefined && pn.point.oy >= (magInfoList[magInfoNum].
                      squashnw.y)) {
223                 percent = (magInfoList[magInfoNum].magnw.oy - pn.point.oy)/
                        magInfoList[magInfoNum].height;
224                 setNewCoordsDiag(pn.point, calcNewCoord(this.nw.point.xChange,
                        destX, percent), null);
225                 pn = pn.north;
226               }
227             }
228             pn = this.sw.south;
229             var destX = this.sw.point.ox;
230             if (this.sw.point.oy == magInfoList[magInfoNum].magse.oy) {
231               pn = this.sw.south;
232               while (pn !== undefined && pn.point.oy <= (magInfoList[magInfoNum].
                      squashse.y)) {
233                 percent = (pn.point.oy - magInfoList[magInfoNum].magse.oy)/
                        magInfoList[magInfoNum].height;
234
235                 setNewCoordsDiag(pn.point, calcNewCoord(this.sw.point.xChange,
                        destX, percent), null);
236                 pn = pn.south;
237               }
238             }
239           }
240         }
241       }
242     }
243   }
```

Separating Axis Theorem

Deze code bekijkt of twee gebieden in de *magInfoList* elkaar gaan overlappen na vergroting. In dit geval worden beide vierkanten uit elkaar gedrukt in de richting waar de overlap het kleinste is. De overlap wordt berekend in de functie *intersecting*.

```
1  function detectCollision() {
2    var collissionsAvoided = false;
3    while (!collissionsAvoided) {
4      collissionsAvoided = true;
5      for (var i = 0; i < magInfoList.length; i++) {
6        for (var j = i + 1; j < magInfoList.length; j++) {
7          var intersectionVector = intersecting(magInfoList[i],magInfoList[j]);
8          if (intersectionVector != null) {
9            if (intersectionVector.x < intersectionVector.y) {
10             var minWidth = min(magInfoList[i].width, magInfoList[j].width);
11             moveMagInfo(magInfoList[i], (intersectionVector.x+minWidth)/2, 0);
12             moveMagInfo(magInfoList[j], -(intersectionVector.x+minWidth)/2, 0);
13           } else {
14             var minHeight = min(magInfoList[i].height, magInfoList[j].height);
15             moveMagInfo(magInfoList[i], 0, (intersectionVector.y+minHeight)/2);
16             moveMagInfo(magInfoList[j], 0, -(intersectionVector.y+minHeight)/2)
                   ;
17           }
18         }
19       }
20     }
21   }
22 }
23
24 function intersecting(X, Y) {
25   var overlapVector = new point(0,0);
26   if (X.magnw.x >= Y.magnw.x && X.magnw.x <= Y.magne.x) {
27     overlapVector.x = (Y.magne.x - X.magnw.x);
28   } else if (X.magnw.x < Y.magnw.x) {
29     if (X.magne.x >= Y.magnw.x) {
30       if ((X.magne.x - Y.magnw.x) > (Y.magne.x - X.magnw.x)) {
31         overlapVector.x = (Y.magne.x - X.magnw.x);
32       } else {
```

```
33          overlapVector.x = -(X.magne.x - Y.magnw.x);
34        }
35      }
36    }
37
38    if (X.magnw.y >= Y.magnw.y && X.magnw.y <= Y.magne.y) {
39      if ((X.magnw.y - Y.magnw.y) > (Y.magsw.y - X.magnw.y)){
40        overlapVector.y = (Y.magsw.y - X.magnw.y);
41      } else {
42        overlapVector.y = -(X.magnw.y - Y.magnw.y);
43      }
44    } else if (X.magnw.y < Y.magnw.y) {
45      if (X.magsw.y >= Y.magnw.y) {
46        overlapVector.y = -(X.magsw.y - Y.magnw.y);
47      }
48    }
49
50    if (overlapVector.x != 0 && overlapVector.y != 0) {
51      return overlapVector;
52    } else {
53      return null;
54    }
55  }
```

Subdivision

Deze code deelt de treemap op in stukken. De parameters *nw* en *se* geven het gebied aan dat in de treemap moet worden gedeeld.

```
1
2  tmTile.prototype.subdivide = function (nw, se) {
3    var north = false;
4    var east = false;
5    var south = false;
6    var west = false;
7    var oneDimNorth = false;
8    var oneDimEast = false;
9    var oneDimSouth = false;
10   var oneDimWest = false;
11
12   if (this.childTiles != null) {
13     for (var i = 0; i < this.childTiles.length; i++) {
14       for (var j = 0; j < this.childTiles[i].length; j++) {
15         this.childTiles[i][j].subdivide(nw, se, begin, end);
16       }
17     }
18   } else {
19     if (nw.y > this.nw.point.y && nw.y < this.sw.point.y) {
20       if (nw.x < this.nw.point.x) { //the north line begins west of the tile
21         if (se.x > this.nw.point.x) {
22           //the north line passes through the west side of the tile
23           north = true;
24         }
25       } else if (nw.x < this.ne.point.x) {
26         north = true;
27       }
28       oneDimNorth = !north;
29     }
30
31     if (se.x > this.nw.point.x && se.x < this.ne.point.x) {
32       if (nw.y < this.nw.point.y) { //the east line begins north of the tile
33         if (se.y > this.nw.point.y) {
34           //the east line passes through the north side of the tile
35           east = true;
36         }
37       } else if (nw.y < this.se.point.y) {
38         east = true;
39       }
40       oneDimEast = !east;
41     }
42
```

```
43      if (se.y > this.nw.point.y && se.y < this.sw.point.y) {
44        if (nw.x < this.nw.point.x) { //the south line begins west of the tile
45          if (se.x > this.nw.point.x) {
46            //the north line passes through the west side of the tile
47            south = true;
48          }
49        } else if (nw.x < this.ne.point.x) {
50          south = true;
51        }
52        oneDimSouth = !south;
53      }
54
55      if (nw.x > this.nw.point.x && nw.x < this.ne.point.x) {
56        if (nw.y < this.nw.point.y) { //the west line begins north of the tile
57          if (se.y > this.nw.point.y) {
58            //the west line passes through the north side of the tile
59            west = true;
60          }
61        } else if (nw.y < this.sw.point.y) {
62          west = true;
63        } else {
64          oneDimWest = true;
65        }
66        oneDimWest = !west;
67      }
68
69      if (north || east || south || west || oneDimNorth || oneDimEast ||
            oneDimSouth || oneDimWest) {
70        this.changedFor = new square(begin, end);
71        if (north && east && south && west) {
72          this.subdivide9(nw.x, nw.y, se.x, se.y);
73          this.childTiles[1][1].squash = true;
74          this.childTiles[1][1].partOfMag = true;
75
76        } else if (north && east && south) {
77          this.subdivide6Ver(nw.y, se.y, se.x);
78          this.childTiles[0][1].squash = true;
79          this.childTiles[0][1].partOfMag = true;
80        } else if (north && east && west) {
81          this.subdivide6Hor(nw.x, se.x, nw.y);
82          this.childTiles[1][1].squash = true;
83          this.childTiles[1][1].partOfMag = true;
84        } else if (north && south && west) {
85          this.subdivide6Ver(nw.y, se.y, nw.x);
86          this.childTiles[1][1].squash = true;
87          this.childTiles[1][1].partOfMag = true;
88        } else if (east && south && west) {
89          this.subdivide6Hor(nw.x, se.x, se.y);
90          this.childTiles[1][0].squash = true;
91          this.childTiles[1][0].partOfMag = true;
92
93        } else if (north && east) {
94          this.subdivide4(se.x, nw.y);
95          this.childTiles[0][1].squash = true;
96          this.childTiles[0][1].partOfMag = true;
97        }  else if (north && west) {
98          this.subdivide4(nw.x, nw.y);
99          this.childTiles[1][1].squash = true;
100         this.childTiles[1][1].partOfMag = true;
101       } else if (east && south) {
102         this.subdivide4(se.x, se.y);
103         this.childTiles[0][0].squash = true;
104         this.childTiles[0][0].partOfMag = true;
105       } else if (south && west) {
106         this.subdivide4(nw.x, se.y);
107         this.childTiles[1][0].squash = true;
108         this.childTiles[1][0].partOfMag = true;
109
110       } else if (north && south) {
111         this.subdivide3Ver(nw.y, se.y);
112         this.childTiles[0][1].squash = true;
```

```
113           this.childTiles[0][1].partOfMag = true;
114         }else if (east && west) {
115           this.subdivide3Hor(nw.x, se.x);
116           this.childTiles[1][0].squash = true;
117           this.childTiles[1][0].partOfMag = true;
118
119         } else if (north) {
120           this.subdivide2Ver(nw.y);
121           this.childTiles[0][1].squash = true;
122           this.childTiles[0][1].partOfMag = true;
123         } else if (east) {
124           this.subdivide2Hor(se.x);
125           this.childTiles[0][0].squash = true;
126           this.childTiles[0][0].partOfMag = true;
127         } else if (south) {
128           this.subdivide2Ver(se.y);
129           this.childTiles[0][0].squash = true;
130           this.childTiles[0][0].partOfMag = true;
131         } else if (west) {
132           this.subdivide2Hor(nw.x);
133           this.childTiles[1][0].squash = true;
134           this.childTiles[1][0].partOfMag = true;
135         } else if (this.partOfMag) {
136           if (oneDimNorth && oneDimSouth) {
137             if (this.squash) {
138               this.subdivide3Ver(nw.y, se.y);
139             }
140           } else if (oneDimEast && oneDimWest) {
141             if (this.squash) {
142               this.subdivide3Hor(nw.x, se.x);
143             }
144           } else if (oneDimNorth ) {
145             if (this.squash) {
146               this.subdivide2Ver(nw.y);
147             }
148           } else if (oneDimEast) {
149             if (this.squash) {
150               this.subdivide2Hor(se.x);
151             }
152           } else if (oneDimSouth) {
153             if (this.squash) {
154               this.subdivide2Ver(se.y);
155             }
156           } else if (oneDimWest) {
157             if (this.squash) {
158               this.subdivide2Hor(nw.x);
159             }
160           }
161         }
162       } else { //check the reverse, check if this Tile is entirely encapsulated
            within the square to subdivide
163         if (this.nw.point.x > nw.x && this.ne.point.x < se.x) {
164           if (this.nw.point.y > nw.y && this.sw.point.y < se.y) {
165             this.squash = true;
166             this.partOfMag = true;
167           }
168         }
169       }
170     }
171 }
```